AD-A198 695

(1

DTIC FILE COPY

DTIC
S ELECTE D
AUG 0 2 1988
H

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

88 [illegible]

-P.-

partial differential equations (co-author) (1)

Abstract. Adaptive methods for ~~PDEs~~ can be viewed as a graph problem. Parallel methods must distribute this graph efficiently among the processors. In doing this, the cost of communication between processors and the structure of the graph must be considered. ~~We~~ divides this problem into two phases: labeling of graph nodes and subsequent mapping of these labels onto processors. ~~We~~ He describes a new form of Gray-code which ~~we~~ calls an *interleaved* Gray-code that allows easy labeling of graph nodes even when the maximal level of refinement is unknown, allows easy determination of nearby nodes in the graph, is completely deterministic, and often (in a well-defined sense) distributes the graph efficiently across a hypercube. The theoretical results are supported by computational experiments on the Connection Machine. (KR)

he

Recursive Mesh Refinement on Hypercubes

W. D. Gropp, I.C.F. Ipsen

Research Report YALEU/DCS/RR-616
March 1988

The work presented in this paper was supported by the Office of Naval Research under contract N00014-85-K-0461, by the Army Research Office under contract DAAL03-86-K-0158, by the Office of Naval Research under contract number N00014-86-K-0310 and the National Science Foundation under contract number DCR 8521451. Part of this work was performed while the second author was in residence at the Computer Science and Systems Division of Harwell Laboratory, United Kingdom. Approved for public release: distribution is unlimited.

1. Introduction

Parallel computing offers the possibility of greatly increased computing power. However, some problems are so large that even enormous parallel computers will not be able to handle them. Such problems include time-dependent partial differential equations (PDEs), which comprise regions with a fine-scale structure as well as regions with a coarse-scale structure. Such problems, solved on a uniform grid, may require on the order of 10^{13} floating operations per time step on a uniform three-dimensional mesh of a thousand points in each direction. The resulting resolution, however, often yields unnecessarily accurate solutions in the coarsely structured regions.

A tremendous amount of work may be saved by adapting the computations to the structure of the PDE at hand. The parallel implementation of such an adaptive method can be considered as the problem of managing a dynamic graph on a (static) network of processors. The properties desired of this graph are that its edges, as much as possible, map to physical processor interconnections, and that changes in the graph do not require a major re-arrangement regarding the assignment of nodes to the processors. Adaptive methods on parallel architectures have received little attention since the graph management problem is hard in general and does not parallelize very well. Our approach rests on the fact that problems suitable for adaptive refinement do *not* require random refinements but exhibit a certain *coherence*: the solution as well as the behavior of any discontinuities (such as shocks) are (nearly) continuous and thus the refinements are *localized*. This can be successfully exploited for operations on data structures as well as maintaining reasonably uniform balance of workload across the processors.

We have made the following choices for the solution of our problem. The initial domain on which a PDE is to be computed is discretised and can be represented as the selective recursive subdivision of a k-dimensional cartesian *grid of cells* (without loss of generality, the maximal number of cells per grid is assumed to be a power of two).

This sort of structure arises from finite difference approximations on adaptive meshes, such as those produced by *Local Uniform Mesh Refinement* (LUMR) [2, 3]. Initially, the k-dimensional grid of cells is *uniform* and function values are computed in each cell of the grid using information from neighboring cells. If there are cells (parents) whose computed values are not accurate enough LUMR establishes a new grid level that contains 2^k cells (children) for each 'inaccurate' cell (i.e. the cell is 'refined'). Function values are then computed for the cells of the finer grids; if the values in a cell are accurate enough they are promoted to the parent cell, otherwise the process is continued recursively. Thus, the refinement process can be viewed as establishing a hierarchy of successively finer grids.

Due to its rich interconnection structure (as much as its commercial availability) the class of hypercube multiprocessors seems the one most suited to efficient handling of a graph management problem. Because of its massive parallelism we selected the Thinking Machines Connection Machine (some of our algorithms will take into account the particular features of this machine). A mesh refinement algorithm may be implemented on the hypercube by assigning the computation of function values in each cell to a certain processor. All processors compute in parallel the function values on their cells; function values on or near a cell boundary are obtained by exchanging information with the processor that contains the cell sharing that boundary. In order to speed up computation a processor can distribute cells resulting from a subdivision to other processors; in this case there is communication between the processor containing the parent and those containing the children.

Our problem is now to assign cells to particular processors in such a way that processor utilization is high (all processors do worthwhile work most of the time), and that processors containing related cells are not too far apart so as to keep communication cost low. These two objectives are conflicting: if all cells are assigned to the same processor communication costs are zero but the load balance could not be worse. Alternatively, high communication costs and overheads may be incurred when trying to maintain a reasonable load balance. In particular, related cells (those that have to communicate with each other) should be in the same processor or in physically connected processors so that communication can proceed without any intermediate processors (which would have to spend time in forwarding information to target processors). The implicit assumption made here is that the of cost of communication between two processors is proportional to their distance, where distance denotes the smallest number of physical communication links that must be traversed to get from one processor to the other.

Because there are, in general, many more cells than processors and the number of grid levels and cells is unpredictable, we divide the process of assigning cells to processors into two stages:

1. Each cell is assigned a *unique* label.
2. Each label is mapped to a processor identifier (id).

The first stage preserves coherence: it is easy to find the labels of siblings, ancestors and descendants of a cell in the hierarchy of grids; once the label has been determined the corresponding processor can be found without much effort. The second stage makes it possible to ensure that related cells are allotted to physically close processors of the hypercube, and that the work load is distributed reasonably across all processors for our applications. Since the labelling strategy is static, a cell can determine the label of any other cell at any level without requiring external information.

Since the processors in a hypercube can be enumerated in such a way that the (binary) identifiers of physically connected processors differ only in one bit (Gray codes) [1], the labels and processor ids are represented as binary numbers. Therefore, the computation times of all our algorithms are measured in operations on bits, and their time complexities are always low-order polynomials in the lengths of the labels. The exact expressions for the computation times depend on the particular implementation and will not be presented here.

The usual method of labelling multi-dimensional grids proceeds by generating labels in such a way that the ith group of *contiguous* bits in the label represents the coordinate with respect to the ith dimension of the grid, and the coordinates associated with each dimension are generated as members of a Gray code sequence [1, 5, 6]. The obvious properties of such a labelling are that adjacent cells are easily mapped to physically interconnected hypercube processors and that each processor can systematically determine the label, and thus the processor, of an arbitrary cell, in particular a neighboring cell. However, it seems to be hard to achieve reasonably uniform processor utilization with a labelling based on a contiguous Gray code when the maximal level of refinement as well as the number of cells are not known in advance.

As a result, for the solution of PDEs on k-dimensional grids, we decided to employ a so-called *interleaved k-dimensional* Gray code that 'scatters' the bits associated with a coordinate: the (ik)th bit in a label represents the coordinate of dimension i in the grid. The length of a label increases with increasing level of refinement. Although interleaved Gray codes superficially resemble Quadcodes [4], the latter do not yield the small communication distances of interleaved Gray codes. In fact, for practical applications, interleaved Gray codes result in essentially constant communication times.

As for the organization of this paper, Section 2 presents the basic properties of Gray codes, and algorithms for labelling cells, determining labels of neighbors in complete and partial one-dimensional hierarchies of grids. The following Section 3 demonstrates the mapping of one-dimensional labels to processor identifiers. Section 4 extends the one-dimensional results to the two- and three-dimensional case by introducing interleaved Gray codes. To assess the quality of our strategy, the model problem of a moving region of refinement is analysed and bounds on processor work load, distance between communicating processors and communication traffic are established. Experimental results are presented in Section 6, followed by a conclusion in Section 7.

2. Labels for Cells in One-Dimensional Grids

This section demonstrates how to generate labels for cells belonging to a single one-dimensional grid, and then for cells belonging to a hierarchy of one-dimensional grids resulting from refinement. The labelling is done in such a way that labels of neighboring cells, and those of parent and children cells differ in only one bit and are easily determined.

Labelling Cells in One Grid

The determination of binary labels $0 \leq \mathcal{L}(i) \leq 2^d - 1$ for the (ordered sequence of) cells $0 \leq i \leq 2^d - 1$ in a one-dimensional grid of 2^d cells is accomplished by means of a Gray code [1] in such a way that labels of successive cells differ in only bit.

Definition 2.1. A d-bit ***binary reflected Gray code***, ***Gray code*** from now on, is an ordered sequence of d-bit binary numbers $\mathcal{L}_d(i)$, $0 \leq i \leq 2^d - 1$, that are recursively generated as follows:

$$\begin{aligned} d = 1 &: \mathcal{L}_1(0) = 0, \quad \mathcal{L}_1(1) = 1 \\ d > 1 &: \mathcal{L}_d(0) = 0\mathcal{L}_{d-1}(0), \quad \cdots, \quad \mathcal{L}_d(2^{d-1} - 1) = 0\mathcal{L}_{d-1}(2^{d-1} - 1), \\ &\quad \mathcal{L}_d(2^{d-1}) = 1\mathcal{L}_{d-1}(2^{d-1} - 1), \quad \cdots, \quad \mathcal{L}(2^d - 1) = 1\mathcal{L}_{d-1}(0). \end{aligned}$$

The ***left neighbor*** of $\mathcal{L}(i)$ is $\mathcal{L}(i-1)$ and the ***right neighbor*** is $\mathcal{L}(i+1)$, where the indices are taken modulo 2^d.

Example. Gray codes for $d = 1, 2, 3$.

$\mathcal{L}_1(0) = 0$	$\mathcal{L}_2(0) = 00$	$\mathcal{L}_3(0) = 000$
$\mathcal{L}_1(1) = 1$	$\mathcal{L}_2(1) = 01$	$\mathcal{L}_3(1) = 001$
	$\mathcal{L}_2(2) = 11$	$\mathcal{L}_3(2) = 011$
	$\mathcal{L}_2(3) = 10$	$\mathcal{L}_3(3) = 010$
		$\mathcal{L}_3(4) = 110$
		$\mathcal{L}_3(5) = 111$
		$\mathcal{L}_3(6) = 101$
		$\mathcal{L}_3(7) = 100$

Remarks.

- The first member of a Gray code sequence is $0 \cdots 0$, and the last one $10 \cdots 0$.
- The Gray code sequence is *cyclic*, i.e. each member of the sequence differs from its successor in exactly one bit, including the last member and the first.

Definition 2.2. $\oplus$ denotes the bitwise *exclusive-or* (XOR) operation:

$$\begin{array}{ccccc} 0 & \oplus & 0 & = & 0 \\ 0 & \oplus & 1 & = & 1 \\ 1 & \oplus & 0 & = & 1 \\ 1 & \oplus & 1 & = & 0, \end{array}$$

while the overbar indicates the complement of a binary number: $\bar{0} = 1$ and $\bar{1} = 0$.

The following two theorems describe the conversion between binary and Gray code representation using simple bit operations. To this end, we start with a lemma that shows that the highest power of two in a number i is also the highest power of two in its Gray code representation $\mathcal{L}_d(i)$.

Lemma 2.1. *A sequence $\mathcal{M}_d$ constitutes a d-bit binary reflected Gray code if and only if*

- *consecutive members of $\mathcal{M}_d$ differ in exactly one bit,*
- *the highest power of two in $\mathcal{M}_d(i)$ agrees with that in i, that is, $\mathcal{M}_d(0) = 0 \cdots 0$, and bit $d-k$ in $\mathcal{M}_d(i)$ is one whenever $2^{k-1} \le i < 2^k$ and $0 < k \le d$.*

Proof. Follows from induction on Definition 2.1. Note that bit 0 in the k-bit Gray code $\mathcal{L}_k(i)$ is one whenever $2^{k-1} \le i < 2^k$, thus bit $d-k$ in the longer sequence $\mathcal{L}_d(i) = \underbrace{0 \cdots 0}_{d-k} \mathcal{L}_k(i)$ is equal to one whenever $2^{k-1} \le i < 2^k$.

Theorem 2.1. *From Binary to Gray Code Representation.*
Let $(i)_2$ denote the d-bit binary representation of a number $0 \le i \le 2^d - 1$ and let $\mathcal{L}$ denote a d-bit Gray code. Then

$$\mathcal{L}(i) = (i)_2 \oplus (\lfloor i/2 \rfloor)_2 \tag{2.1}$$

where $(\lfloor i/2 \rfloor)_2$ also has a d-bit representation.

Proof. For any two consecutive numbers $(i)_2$ and $(i+1)_2$ there exists a $k \ge 1$ so that each of the k trailing bits in $(i)_2$ differs from that in $(i+1)_2$; for instance, each of the trailing two bits in $(5)_2 = 101$ differs from its counter part in $(6)_2 = 110$. Let

$$\begin{aligned} (i)_2 &= i_0 \cdots i_{d-k-1}\, i_{d-k-2} \cdots i_{d-1} \\ (i+1)_2 &= i_0 \cdots i_{d-k-1}\, \bar{i}_{d-k-2} \cdots \bar{i}_{d-1} \end{aligned}$$

so that

$$\begin{aligned} (\lfloor i/2 \rfloor)_2 &= 0\, i_0 \cdots i_{d-k-1}\, i_{d-k-2} \cdots i_{d-2} \\ (\lfloor (i+1)/2 \rfloor)_2 &= 0\, i_0 \cdots i_{d-k-1}\, \bar{i}_{d-k-2} \cdots \bar{i}_{d-2}. \end{aligned}$$

Let

$$\mathcal{M}(i) = (i)_2 \oplus (\lfloor i/2 \rfloor)_2.$$

Because of $i_{j-1} \oplus i_j = \bar{i}_{j-1} \oplus \bar{i}_j$ for $d-k-2 \ge j \ge d-2$ it follows that $\mathcal{M}(i)$ and $\mathcal{M}(i+1)$ differ in bit $d-k-2$. Thus, two consecutive numbers in the sequence $\mathcal{M}(i)$ differ in exactly one bit, and $\mathcal{M}(0) = 0 \cdots 0$. Furthermore, by definition, the highest power of two in $\mathcal{M}(i)$ is the same as that of i. Applying Lemma 2.1 one can conclude that $\mathcal{M}$ is the binary reflected Gray code $\mathcal{L}_d$.

Example. Illustration of Theorem 2.1 for the case $d = 3$.

$(i)_2$	$\oplus$	$(\lfloor i/2 \rfloor)_2$	$=$	$\mathcal{L}(i)$
000		000		000
001		000		001
010		001		011
011		001		010
100		010		110
101		010		111
110		011		101
111		011		100

Thus, the label of cell i in a one-dimensional grid of 2^d cells, $0 \le i \le 2^d - 1$, is given by $\mathcal{L}(i)$.

Theorem 2.2. *From Gray Code to Binary Representation.*
The binary representation $(i)_2$ of the position of a binary number $l_0 \cdots l_{d-1}$ within the d-bit Gray code sequence $\mathcal{L}$, so that $\mathcal{L}(i) = l_0 \cdots l_{d-1}$, can be obtained from

$$\begin{aligned} i_0 &= l_0 \\ i_k &= i_{k-1} \oplus l_k, \qquad 1 \le k \le d-1. \end{aligned} \tag{2.2}$$

Proof. The proof follows from (2.1): $(i)_2 = (\lfloor i/2 \rfloor)_2 \oplus \mathcal{L}(i)$.

Remark. If $\mathcal{L}_d(i)$ is at the ith position in the d-bit Gray code sequence $\mathcal{L}_d$ then it is also at the ith position in any d'-bit Gray code sequence with $d' > d$.

Determining the Labels of Neighboring Cells in One Grid

Now that we know how to assign a unique label to each cell in a one-dimensional grid, we want to quickly find the labels of the left neighbor and the right neighbor of a cell; here, 'left' means preceding in the Gray code sequence and 'right' succeeding. At first, we introduce a function that detects the parity of bits equal to one in a binary number.

Definition 2.3. Denote by $\mathcal{X}(i)$ the function that XORs all bits of the binary number $(i)_2 = i_0 \cdots i_{d-1}$ and that is equal to one if $(i)_2$ contains an odd number of bits equal to one.

$$\mathcal{X}(i) = \begin{cases} i_0 & \text{if } d = 1 \\ \mathcal{X}(i_0 \cdots i_{d-2}) \oplus i_{d-1} & \text{if } d > 1. \end{cases}$$

Lemma 2.2. *Finding One Neighbor of a Cell.*
If $\mathcal{L}(i)$ has an even number of ones then its successor $\mathcal{L}(i+1)$ in the Gray code sequence is obtained by complementing the rightmost bit in $\mathcal{L}(i)$. If $\mathcal{L}(i)$ has an odd number of ones then its predecessor $\mathcal{L}(i-1)$ in the Gray code sequence is obtained by complementing the rightmost bit in $\mathcal{L}(i)$. Formally, if $\mathcal{L}(i) = i_0 \cdots i_{d-2} i_{d-1}$ then

$$\begin{aligned} \mathcal{L}(i+1) &= i_0 \cdots i_{d-2}\bar{i}_{d-1}, \quad \text{if} \quad \mathcal{X}(\mathcal{L}(i)) = 0 \\ \mathcal{L}(i-1) &= i_0 \cdots i_{d-2}\bar{i}_{d-1}, \quad \text{if} \quad \mathcal{X}(\mathcal{L}(i)) = 1 \end{aligned}$$

Proof. If $\mathcal{L}(i)$ has an even number of ones then i is even, since $\mathcal{L}(0) = 0 \cdots 0$ has an even number of ones and successive Gray code members differ in exactly one bit. If i is even then the last bit of $(i)_2$ is zero, thus $(i)_2$ and $(i+1)_2$ differ in their last bit; also $\lfloor i/2 \rfloor = \lfloor (i+1)/2 \rfloor$. With (2.1) it follows that $\mathcal{L}(i)$ and $\mathcal{L}(i+1)$ differ in their last bit.

Lemma 2.3. *Finding the Other Neighbor of a Cell.*
If $\mathcal{L}(i)$ has an even (odd) number of ones, then its predecessor $\mathcal{L}(i-1)$ (its successor $\mathcal{L}(i+1)$) in the Gray code sequence is obtained by complementing the bit preceding the rightmost one in $\mathcal{L}(i)$. Formally, if $\mathcal{L}(i) = i_0 \cdots i_{k-1} i_k 10 \cdots 0$ then

$$\begin{aligned} \mathcal{L}(i-1) &= i_0 \cdots i_{k-1}\bar{i}_k 10 \cdots 0, \quad \text{if} \quad \mathcal{X}(\mathcal{L}(i)) = 0 \\ \mathcal{L}(i+1) &= i_0 \cdots i_{k-1}\bar{i}_k 10 \cdots 0, \quad \text{if} \quad \mathcal{X}(\mathcal{L}(i)) = 1 \end{aligned}$$

In addition, there are two special cases: the left neighbor of 0 $\cdots$ 0 *is* 10 $\cdots$ 0, *and the right neighbor of* 10 $\cdots$ 0 *is* 0 $\cdots$ 0.

Proof. By induction using Definition 2.1 of Gray codes. Assuming the statement is true for the sequence $\mathcal{L}_d$, then it is also true for the sequence $0\mathcal{L}_d$ and for the reversed sequence $1\mathcal{L}_d$. At the boundary of the two sequences one has: $\mathcal{L}_d(2^d-1) = 10\cdots0$ so $\mathcal{X}(0\mathcal{L}_d(2^d-1)) = 1$, and its right neighbor is $1\mathcal{L}_d(2^d-1) = 110\cdots0$.

Labelling Cells in a Hierarchy of Grids

A *complete hierarchy* or *complete refinement* of one-dimensional grids represents a sequence of grids where each grid contains twice as many cells as its predecessor.

Definition 2.4. In a *complete hierarchy of grids*, the initial grid at *level* 0 consists of one cell, and a grid at *level* k consists of 2^k cells. Each cell (parent) at *level* k gives rise to two cells (children) in the next finer grid at *level* $k+1$. The terms *level* and *grid level* will be used interchangeably.

The assignment of labels to cells should be conducted in such a fashion that the label of a cell differs in exactly one bit from that of its two neighbors in the same grid, and in one bit from that of its parent. This is accomplished by introducing a hierarchy of Gray code sequences: a grid at level k is associated with a Gray code sequence of length 2^k. The construction of such a hierarchy becomes clear from the next lemma that gives an alternative characterization of Gray codes. As for notation, a full stop in a binary sequence denotes concatenation.

Lemma 2.4. *If the sequence* $\mathcal{L}(0)\cdots\mathcal{L}(2^d-1)$ *represents a d-bit Gray code then the sequence*

$$\mathcal{M}(2i) = \mathcal{L}(i).\mathcal{X}(\mathcal{L}(i))$$
$$\mathcal{M}(2i+1) = \mathcal{L}(i).\overline{\mathcal{X}}(\mathcal{L}(i)), \qquad 0 \le i \le 2^d-1$$

represents a $(d+1)$*-bit Gray code.*

Proof. Since the sequence $\mathcal{L}(0)\cdots\mathcal{L}(2^d-1)$ constitutes a Gray code all members in $\mathcal{M}(0)\cdots\mathcal{M}(2^{d+1}-1)$ are different. It remains to show that the neighboring elements differ in one bit. $\mathcal{M}(2i) = \mathcal{L}(i).\mathcal{X}(\mathcal{L}(i))$ and $\mathcal{M}(2i+1) = \mathcal{L}(i).\overline{\mathcal{X}}(\mathcal{L}(i))$ differ in their rightmost bit.

If $\mathcal{L}(i)$ and $\mathcal{L}(i+1)$ differ in one bit then $\mathcal{X}(\mathcal{L}(i)) \ne \mathcal{X}(\mathcal{L}(i+1))$ hence $\overline{\mathcal{X}}(\mathcal{L}(i)) = \mathcal{X}(\mathcal{L}(i+1))$ so that $\mathcal{M}(2i+1) = \mathcal{L}(i).\overline{\mathcal{X}}(\mathcal{L}(i))$ and $\mathcal{M}(2i+2) = \mathcal{L}(i+1).\mathcal{X}(\mathcal{L}(i+1))$ differ in one bit. It is now clear that the sequence $\mathcal{M}$ fulfills all properties of Lemma 2.1 and hence represents a Gray code.

Example.

$\mathcal{L}(0) = 00$,	$\mathcal{X}(\mathcal{L}(0)) = 0$,	$\mathcal{M}(0) = 00.0$	$= 000$
		$\mathcal{M}(1) = 00.\overline{0}$	$= 001$
$\mathcal{L}(1) = 01$,	$\mathcal{X}(\mathcal{L}(1)) = 1$,	$\mathcal{M}(2) = 01.1$	$= 011$
		$\mathcal{M}(3) = 01.\overline{1}$	$= 010$
$\mathcal{L}(2) = 11$,	$\mathcal{X}(\mathcal{L}(2)) = 0$,	$\mathcal{M}(4) = 11.0$	$= 110$
		$\mathcal{M}(5) = 11.\overline{0}$	$= 111$
$\mathcal{L}(3) = 10$,	$\mathcal{X}(\mathcal{L}(3)) = 1$,	$\mathcal{M}(6) = 10.1$	$= 101$
		$\mathcal{M}(7) = 10.\overline{1}$	$= 100$

Due to Lemma 2.4 each member $\mathcal{L}(i)$ of the old Gray code can independently generate two members $\mathcal{M}(2i)$ and $\mathcal{M}(2i+1)$ of the new Gray code. Thus, it is possible to determine the labels of the two children of a cell solely from its own label – without any knowledge of other cells' labels. Note, that the leftmost cell on each level has label $0 \cdots 0$ while the rightmost cell has label $10 \cdots 0$.

In order not to have to deal with labels of different lengths, it is assumed that each label consists of d bits, where d is large enough to accommodate all possible refinements. Hence, the leading k bits in labels of cells belonging to a level k grid form a k-bit Gray code and the remaining bits are zero. This assumption is by no means required for the theory to work out, but is helpful for purposes of implementation.

Example. Generation of labels for a complete hierarchy of four grids, $d = 3$.

Level 0 :	000							
Level 1 :	000							100
Level 2 :	000			010	110			100
Level 3 :	000	001	011	010	110	111	101	100

To avoid confusion, we now formally define when two cells are neighbors within the same grid.

Definition 2.5. Two cells are neighbors at level $k > 0$ if their labels are of the form $c_0 \cdots c_{k-1}0 \cdots 0$ and are adjacent in the k-bit reflected Gray code sequence.

Determining the Labels of Neighboring Cells in a Complete Hierarchy of Grids

Once a label has occured on a level it also occurs on all lower levels. That is, a cell has in each lower level grid one ancestor with the same label. To facilitate determination of neighboring labels in a hierarchy of grids we make the following definition.

Definition 2.6. In a hierarchy of d grids, a cell C with label $\mathcal{L}(C) = c_0 \cdots c_{d-1}$ is said to *originate* at grid level k, $1 \leq k \leq d$, if k is the largest number so that $c_{k-1} = 1$ and $c_k = \cdots = c_{d-1} = 0$; the cell with label $0 \cdots 0$ originates at level 0.

For example, $10\ldots0$ originates at level 1 and $10010\ldots0$ originates at grid level 4. Hence, in the last example, for instance, 000 has as right neighbors 100 on grid level 1, 010 on grid level 2, and 001 on grid level 3. If a cell originates at level k, it has neighbors on grid level k and on all grid levels $> k$.

A cell C may determine the labels of all of its neighbors by means of the following strategy. Suppose, the label of C contains an even number of ones, i.e. $\mathcal{X}(\mathcal{L}(C)) = 0$, and that it originates at level k, i.e. $\mathcal{L}(C) = c_0 \cdots c_{k-2}10 \cdots 0$.

- Right Neighbors of C.
  By Lemma 2.2, C has a right neighbor R_k at level k whose label is $\mathcal{L}(R_k) = c_0 \cdots c_{k-2}00 \cdots 0$; and cells R_i with labels $\mathcal{L}(R_i) = c_0 \cdots c_{k-2}1\underbrace{0 \cdots 0}_{i-(k+1)}10 \cdots 0$ are right neighbors of C on level i, $i > k$. Thus, C can compute the labels of its right neighbors on lower levels i by setting the ith bit in its own label equal to 1. For example, 000 has at level 2 a left child with the same label, and a right child with the new label 010, hence 000 has 010 as its right neighbor on level 2.
- Left Neighbors of C.
  By Lemma 2.3, the left neighbor L_k of C on level k has label $\mathcal{L}(L_k) = c_0 \cdots c_{k-3}\bar{c}_{k-2}10 \cdots 0$. It must have an odd number of ones, i.e. $\mathcal{X}(\mathcal{L}(L_k)) = 1$, so the rightmost descendants L_i of L_k inherit this label by Lemma 2.4. Therefore, L_k is the left neighbor of C on all levels. For example, 110 on level 2 has as left neighbor 010 on levels 2 and 3.

Cells C whose labels contain an odd number of ones are treated similarly.

Determining the Labels of Neighboring Cells in a Partial Hierarchy of Grids

In a *partial refinement* or *partial hierarchy* of one-dimensional grids not necessarily every cell is refined, that is not every cell gives rise to two cells on the next level; and labels of neighbors in a partial refinement can differ by more than one bit.

Example. Labels in a partial hierarchy of four grids, $d = 3$.

Level 0 :	000			
Level 1 :	000			100
Level 2 :	000	110		100
Level 3 :	000	110	111	100

Comparing this with the preceding example one sees that cell 000 is not further refined at level 2 and cell 100 is not refined at level 3. Furthermore, 110 has as its right neighbor 111 on level 3; but its left neighbor is 000 which has not been refined after level 1. Hence, 110 and its left neighbor 000 differ in two bits.

The algorithm for finding the neighbors of a cell C in such a refinement is based on the principal property of Local *Uniform* Mesh Refinement: each cell (excepting $0 \cdots 0$) has a sibling on the level on which it originates. Suppose again that cell C has an even number ones in its binary label, i.e. $\mathcal{X}(\mathcal{L}(C)) = 0$, and that it originates at level k, i.e. $\mathcal{L}(C) = c_0 \cdots c_{k-2}10 \cdots 0$.

- Right Neighbors of C.
  By Lemma 2.2, C has a right neighbor R_k at level k whose label is $\mathcal{L}(R_k) = c_0 \cdots c_{k-2}00 \cdots 0$. Those existing cells R_i with labels $\mathcal{L}(R_i) = c_0 \cdots c_{k-2}1\underbrace{0 \cdots 0}_{i-(k+1)}10 \cdots 0$ are right neighbors of C on level i, $i > k$.
- Left Neighbors of C.
  If C has a left neighbor L_k on level k then by Lemma 2.3 its label is $\mathcal{L}(L_k) = c_0 \cdots c_{k-3}\bar{c}_{k-2}10 \cdots 0$, and all left neighbors of C on lower levels have this label. If L_k does not exist then one can reconstruct the ancestors of L_k until an existing one is found (this is done by simply reversing the process for generating labels of children, Lemmata 2.2 and 2.3).

Again, cells C whose labels contain an odd number of ones are treated similarly.

Notions Useful for Applications

The next definition distinguishes those cells in computational problems that communicate frequently with each other.

Definition 2.7. In a (partial) hierarchy of d grids, the *closest left (right) neighbor* N of a cell C at level k is the closest left (right) cousin without any children. If the hierarchy is complete, the closest left and right neighbors of a cell are the left and right neighbors at level $d - 1$.

In the example above, the closest left neighbor of 110 is 000, and its closest right neighbor is 111. The closest left neighbor of 100 is 111. We assume that a cell knows about the status of refinement in its immediate vicinity, i.e. it knows, at least, the levels at which its closest left and right neighbors originate (the label of a closest neighbor can be easily determined from its level, for instance, by means of the alternate characterization of Gray codes in Lemma 2.4).

For purposes of *assessing the workload per processor*, only those cells with no children are considered to be instantiated; that is, if a parent and a child have the same label they are viewed as one cell.

3. Mapping Labels of One-Dimensional Grids to Processors

The previous section showed how to assign a unique label to each cell in a hierarchy of grids. Now we want to map the labels to processors in the hypercube such that the identifier (id) of a processor can be easily computed from the label of the cell, and cells whose labels differ in one bit are assigned to physically neighboring processors.

Definition 3.1. A hypercube multiprocessor of dimension $p \geq 0$ contains 2^p processors where the processors can be identified by p-bit binary ids so that any two processors are physically connected whose ids differ in 1 bit.

Consequently, processors in a hypercube of dimension p can be enumerated according to a binary reflected Gray code sequence of length 2^p. Topological properties of hypercube graphs and their embeddings can be found in [1, 5, 6].

The assignment of labels to processor ids is easy if a grid of 2^d cells is mapped to a hypercube of dimension p where $p \geq d$: the cells in this case occupy processors in a subcube of dimension d, and the label of a cell is at the same time the id of a processor.

The more interesting case occurs when the number 2^d of cells exceeds the number of processors in the hypercube, and when grids are only partially refined. The processor to which a cell C is assigned is denoted by $\mathcal{P}(C)$. Consider a hierarchy of $p+l$ grids where the finest grid contains 2^l more cells than there are processors, $1 \leq l \leq p$. Hence the label of each cell C has length $d = p+l$. Our strategy for mapping labels of length $p+l$, $l \leq p$, to p-dimensional hypercubes can then be defined as follows: a cell C with label $\mathcal{L}(C) = c_0 \cdots c_{p-1}.c_p \cdots c_{p+l-1}$, $1 \leq l \leq p$, is assigned to the processor with id

$$\begin{aligned}\mathcal{P}(C) &= \{c_0 \oplus c_p\}\{c_1 \oplus c_{p+1}\} \cdots \{c_{l-1} \oplus c_{p+l-1}\}.c_l \cdots c_{p-1} \\ &= \{c_0 \cdots c_{p-1}\} \oplus \{c_p \cdots c_{p+l-1}0 \cdots 0\}.\end{aligned}$$

Remarks.

- Since $0 \cdots 0$ is the identity with respect to XOR, trailing zeros behind the fullstop do not change the processor id. Hence, one child always occupies the same processor as its parent and the other child is assigned to an adjacent processor.
- Cells whose labels differ in 1 bit are assigned to physically connected processors; thus, the processor of a cell contains the parent or the sibling, or it is physically connected to a processor that contains the parent or the sibling.
- The mapping is consistent: when $d < p$ and the number of processors in the hypercube exceeds the number of cells in the finest grid then $\mathcal{P}(C) = \mathcal{L}(C)$ for any cell C.
- We consider here only the case where the number of refinements does not exceed the dimension of the hypercube ($l \leq p$). This is sufficient for massively parallel architectures like the Connection Machine.

Example. Consider the case $p = 4$: a 16 processor hypercube, where each processor initially contains one cell. Assume that only a single cell C is further refined. Let C have label $\mathcal{L}(C) = 0100.00$, so it occupies processor $\mathcal{P}(C) = 0100$ (here the fullstop separates the leading p bits from the remaining bits). Refining C results in two children L and R with $\mathcal{L}(L) = 0100.10$ and $\mathcal{L}(R) = 0100.00$, The respective processors assigned to cells L and R are $\mathcal{P}(L) = 1100$ and $\mathcal{P}(R) = 0100$. Note that R occupies the processor of its parent while L is assigned to an adjacent processor. Now, L and R can in turn be refined. The labels of the two children of L are 0100.10 and 0100.11, they are allocated to processors $\mathcal{P}(0100.10) = 1100$ and $\mathcal{P}(0100.11) = 1000$. Similarly, R's children 0100.01 and 0100.00

are assigned to the respective processors $\mathcal{P}(0100.01) = 0000$ and $\mathcal{P}(0100.00) = 0100$. Thus, all four descendants of C occupy different processors.

In general, if the origin of the refinement is a single cell C with $\mathcal{L}(C) = c_0 \cdots c_{p-1}.0 \cdots 0$ then the labels of all cells on level $d = p + l$ are identical in their leading p bits, and form a Gray code in their trailing l bits, $1 \leq l \leq p$; i.e. they are of the form $c_0 \cdots c_{p-1}.x_0 \cdots x_{l-1}$. The processor id of each cell has the form

$$\{c_0 \cdots c_{p-1}\} \oplus \{x_0 \cdots x_{l-1}0 \cdots 0\}.$$

Hence one of the two arguments of the XOR function is the same for all cells. Since the XOR function with a constant argument is bijective each cell is assigned a distinct processor, and in addition adjacent cells occupy adjacent processors. After $l = p$ refinements each processor contains one descendant of cell C. Thus, processor allocation and utilization are uniform when the origin of refinement is single cell.

Now consider the situation when two adjacent cells C_1 and C_2 constitute the origin of refinement, say $p = 4$ and $\mathcal{L}(C_1) = 0100.0$ and $\mathcal{L}(C_2) = 1100.0$. The children of C_1 are 0100.0 on processor 0100 and 0100.1 on processor 1100, the children of C_2 are 1100.0 on processor 1100 and 1100.1 on processor 0100. Obviously the children of the cells occupy each others processors instead of spreading the load to different processors. This interference occurs because XORing affects exactly that bit in which C_1 and C_2 differ (the leading bit), and this difference gets lost because each child of C_1 has the same trailing bit as a child of C_2. In general, if two cells differ in bit i, $i \geq 0$, then at the $(i+1)$st refinement each descendant of one cell will be assigned to the same processor as a descendant of the second cell.

Yet, instead of starting XORing with the bit in position 0 and working towards the right, one could also start XORing with the bit in position $p - 1$ and work to the left. In the latter case, however, chances of interference are much greater because of the following argument: in a d-bit Gray code sequence there are 2^{d-1} adjacent members that differ in their last bit, hence XORing on the last bit would result in a 50% chance of interference. In contrast, there are only two pairs of adjacent members that differ in their leading bit so the likelyhood of interference would be one in 2^{d-1}.

One can always construct cases where our strategies will deliver the worst processor utilization. However, for our applications such as problems with few regions of concentrated refinement, we can give upper bounds on the processor load imbalance, the physical distance to neighboring cells and the communication traffic. This is done for problems containing a moving region of refinement in Section 5. One could devise dynamic strategies that give preference to XORing those bits that are identical in all cells to be refined. We will not consider those strategies here as they are not predictable or static, and are likely to require a considerable amount of global information.

4. Labels and Mappings for Two-Dimensional Grids

The labels of the cells in a $2^d \times 2^d$ grid are to be assigned in such a way that labels of the immediately neighboring cells differ in exactly one bit. By 'immediate neighbors' of a cell we mean the north, east, south and west neighbors. Denote the cell in row x and column y of the grid by (x, y), $0 \leq x, y \leq 2^d - 1$. Gray codes for multi-dimensional grids are usually constructed by allotting a contiguous block of bits to each dimension [1, 6]. We use an *interleaved two-dimensional Gray code*, *interleaved Gray code* for short, to determine the labels: If $\mathcal{L}(x) = x_0x_1 \cdots x_{d-1}$ and $\mathcal{L}(y) = y_0y_1 \cdots y_{d-1}$ are the respective Gray codes for the first and second coordinate of cell (x, y) then its label is $\mathcal{L}(x, y) = x_0y_0x_1y_1 \cdots x_{d-1}y_{d-1}$. Obviously, interleaved Gray codes can also be used to label higher-dimensional grids.

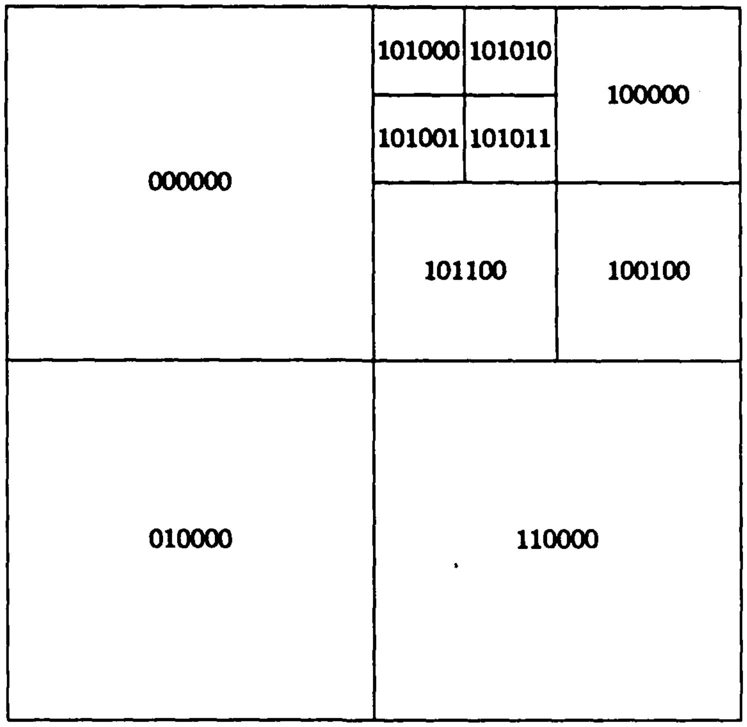

Figure 1: Partial hierarchy of three two-dimensional grids.

Example. Labels for a 4×4 grid.

0000	0010	1010	1000
0001	0011	1011	1001
0101	0111	1111	1101
0100	0110	1110	1100

Thus, if x constitutes the horizontal coordinate axis and y the vertical one then the x coordinates form a horizontal Gray code and the y coordinates from a vertical Gray code. All the facts that have been established for one-dimensional grids naturally carry over to two-and higher-dimensional grids by applying the one-dimensional strategy to each coordinate separately. Also, labels of multi-dimensional grids are mapped to processors by mapping each coordinate separately according to the one-dimensional strategy (this is possible because grids of any dimension can be embedded into a suitably sized hypercube [5, 6]).

For example, when constructing a hierarchy of two-dimensional grids, level 1 contains cells

$00.0 \cdots 0$	$01.0 \cdots 0$
$10.0 \cdots 0$	$11.0 \cdots 0$

and a cell with label $\mathcal{L}_k(x, y) = x_0y_0 \cdots x_{k-1}y_{k-1}.0 \cdots 0$ at level k gives rise to the following four cells at level $k+1$

$\mathcal{L}(2x, 2y) = L.X_xX_y.0 \cdots 0$	$\mathcal{L}(2x+1, 2y) = L.\overline{X}_xX_y.0 \cdots 0$
$\mathcal{L}(2x, 2y+1) = L.X_x\overline{X}_y.0 \cdots 0$	$\mathcal{L}(2x+1, 2y+1) = L.\overline{X}_x\overline{X}_y.0 \cdots 0$

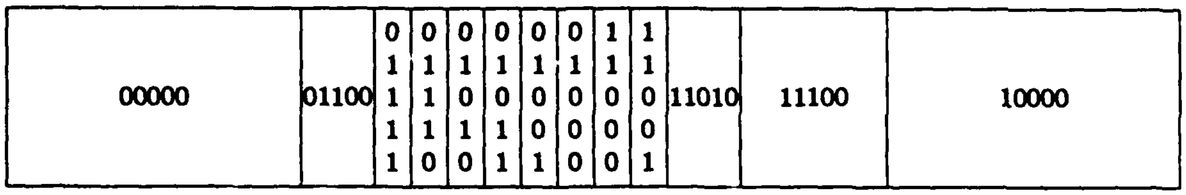

Figure 2: Region of refinement in the 11th position, where $p = 3$ and $l = 2$.

where $L = x_0y_0 \cdots x_{k-1}y_{k-1}$, $X_x = \mathcal{X}(x_0 \cdots x_{k-1})$ and $X_y = \mathcal{X}(y_0 \cdots y_{k-1})$.

As a consequence, labels of cells with the same parent differ in at most two bits, and the label of a cell differs from that of a child by at most two bits. At level k, the horizontal coordinates of a $2^k \times 2^k$ grid form a Gray code sequence of length 2^k, and so the vertical coordinates. In a partial hierarchy of two-dimensional grids, a cell can have more than four neighbors, see Figure 1.

To generate labels for a rectangular $2^k \times 2^{d-k}$ grid consider first a wide grid, i.e. $0 < k < d/2$. In this case only the leading $2k$ bits of the label form an interleaved Gray code and the remaining bits account for the width of the grid: for $0 \leq x \leq 2^k - 1$, $0 \leq y \leq 2^{d-k} - 1$, cell (x, y) is given label $x_0y_0 \cdots x_{k-1}y_{k-1}y_k \cdots y_{d-k-1}$ where $x_0 \cdots x_{k-1} = \mathcal{L}(x)$ and $y_0 \cdots y_{d-k-1} = \mathcal{L}(y)$. The case $k > d/2$ of a tall grid is treated similarly. For simplicity of presentation it thus suffices to restrict the discussion to square grids.

5. Analysis of a Model Problem

Here we consider the problem of one region of refinement that moves across a domain. We chose the moving region of refinement problem not only because of its resemblance to many practical applications, but also because it allows us to examine different locations of intensive refinement within the domain and so to be able to take into account as many cases as possible. At first, we consider a one-dimensional domain. As we show later, the analysis of one moving region of refinement in a two- or three-dimensional domain can be easily reduced to the analysis of the one-dimensional problem.

The One-Dimensional Model

A region of refinement in a one-dimensional domain represents a set of contiguous cells belonging to the grid at level $p + l$, see Figure 2.

Since there is only one region of refinement, the area outside the region of refinement can comprise no two cells that are neighbors on the same level (that is, the cells become 'larger' with increasing distance from the region of refinement, and outside the region of refinement there can be no two cells of the same size). Given a p-dimensional hypercube, we assume at first that the region of refinement consists of 2^p cells belonging to the grid at level $l + p$ (as will be shown later, this assumption is not restrictive). There are two types of cells, ***refined cells*** representing the region of refinement and ***outside cells*** representing the area outside the refinement; due to the natural rules for the refinement, the outside cells are cousins (siblings of ancestors) of the refined cells. Figure 2 shows refined and outside cells for one position of a moving region of refinement.

Communication can take place

- between a cell and its parent, e.g. between 01011 and 01010 in Figure 2;
- between two neighboring refined cells, e.g. between 01000 and 11000 in Figure 2;

- between a refined (boundary) cell and a neighboring outside cell, e.g. between 01111 and 01100 in Figure 2;
- between two neighboring outside cells, e.g. between 11100 and 10000 in Figure 2.

Sections 2 and 3 showed how to determine labels and processors for the parent of a cell, and that the parent is either situated on the same processor as the cell or on a physically connected processor. We will now show that the labels of any two neighboring cells in the above situations differ in no more than two bits. That is, the communication of two such cells involves at most one intermediate processor.

Processor Distance of Neighboring Cells in the One-Dimensional Model

From Sections 2 and 3 we know how to obtain the label and the processor for neighboring cells within the refinement; neighbors within the refinement are always situated on physically connected processors. Outside the refinement one has to deal with a partial hierarchy but in this case there is a simple procedure for finding the labels of all outside cells. To this end we distinguish the two boundary cells of the refinement.

Definition 5.1. For a refinement of 2^p cells belonging to a grid at level $p+l$, $l \geq 1$, C_{start} denotes the leftmost, leading cell and C_{end} the rightmost, trailing cell of the refinement.

Two cases can occur: either the first cell of the region of refinement, C_{start}, has an even number of ones or it has an odd number of ones. The case where it has an odd number of ones, $\mathcal{X}(C_{start}) = 1$, can be dispensed with as follows. If $\mathcal{X}(C_{start}) = 1$ then by Lemma 2.2 it has a left sibling L, so L is the cell outside the region of refinement adjacent to C_{start}. Since the labels of C_{start} and L differ in the rightmost bit, their processor ids differ in 1 bit, and C_{start} and L are allocated to physically connected processors. As the region of refinement is of length 2^p one has $\mathcal{X}(C_{end}) = 0$ if $\mathcal{X}(C_{start}) = 1$. In this case, C_{end} has a right sibling R whose label differs from that of C_{end} in the rightmost bit, by Lemma 2.2, and C_{end} and R are assigned to physically connected processors. Hence, if $\mathcal{X}(C_{start}) = 0$ then C_{start} is the leftmost cell in the grid at level $p+l$, and otherwise L is. As the remaining two situations: communication between two outside cells, and communication between a refinement (boundary) cell and an outside cell when $\mathcal{X}(C_{start}) = 1$ can be dealt with in one, it suffices to consider refinements for which $\mathcal{X}(C_{start}) = 0$.

Lemma 5.1. *Let C be a cell to the left of the region of refinement or C_{start}, and $\mathcal{X}(C) = 0$. The label of the left neighbor L of C is obtained by complementing in $\mathcal{L}(C)$ the bit with the rightmost one and the bit preceding it. Formally, if*

$$\mathcal{L}(C) = c_0 \cdots c_{k-3}c_{k-2}10 \cdots 0 \neq 0 \cdots 0$$

then $\mathcal{L}(L) = c_0 \cdots c_{k-3}\bar{c}_{k-2}00 \cdots 0$, and $\mathcal{X}(L) = 0$.

Proof. Let C originate at level k, i.e. $\mathcal{L}(C) = c_0 \cdots c_{k-3}c_{k-2}10 \cdots 0$. Then the parent P of C has the label $\mathcal{L}(P) = c_0 \cdots c_{k-3}c_{k-2}00 \cdots 0$, and $\mathcal{X}(P) = 1$. By Lemma 2.2 P has a left sibling L with $\mathcal{L}(L) = c_0 \cdots c_{k-3}\bar{c}_{k-2}00 \cdots 0$, and $\mathcal{X}(L) = 0$. As there is only one region of refinement and L is outside the refinement it is not further refined, so L is the left neighbor of C.

Hence, starting with C_{start} and proceeding towards the left one can successively find the left neighbors of all cells to the left of the refinement. If C_{start} consists of an even number of ones then so do all cells to the left of it by Lemma 5.1, and the leftmost cell in the domain is $0 \cdots 0$. Similar facts can be established for the cells to the right of the refinement.

Lemma 5.2. *Let C be a cell to the right of the refinement or C_{end}, and $\mathcal{X}(C) = 1$. The label of the right neighbor R of C is obtained by complementing in $\mathcal{L}(C)$ the bit with the rightmost one and the bit preceding it. Formally, if*

$$\mathcal{L}(C) = c_0 \cdots c_{k-3}c_{k-2}10 \cdots 0 \neq 10 \cdots 0$$

then $\mathcal{L}(R) = c_0 \cdots c_{k-3}\bar{c}_{k-2}00 \cdots 0$, *and* $\mathcal{X}(R) = 1$.

Proof. Analogous to the one of Lemma 5.1.

Hence, starting with C_{end} and advancing towards the right one can successively find the right neighbors of all cells to the right of the region of refinement. If C_{end} consists of an odd number of ones then so do all cells to the right of it by Lemma 5.2, and the rightmost cell in the domain is $10 \cdots 0$.

Independently of the value of $\mathcal{X}(C_{start})$, the number of cells to the left of the region of refinement can be obtained by determining the position j of $\mathcal{L}(C_{start})$ in the Gray code sequence G_{p+l} so that $G_{p+l}(j) = \mathcal{L}(C_{start})$ using Lemma 2.2, and then counting the number of ones in the binary representation of j. For example, in Figure 2, $p + l = 5$ and $\mathcal{L}(C_{start}) = 01111$, so $01111 = G_5(10)$ and $j = (10)_{10} = (01010)_2$. This means, that C_{start} is preceded by two cells, namely 00000 and 01100. In general, the number of cells to the left of the refinement cannot exceed $p + l$. One can get the number of cells to the right of the refinement similarly by finding i such that $G_{p+l}(i) = \mathcal{L}(C_{end})$ and counting the ones in the binary representation of $2^{p+l} - 1 - i$. In the above example $\mathcal{L}(C_{end}) = 11001$, so $11001 = G_5(17)$ and $2^5 - 1 - i = (14)_{10} = (01110)_2$. Thus, C_{end} is succeeded by three cells: 11010, 11100 and 10000.

The total number of cells to the left and right can be computed by noticing that $i = j + 2^p - 1 \bmod 2^{p+l}$, where the refinement is of length 2^p. For the case $i < 2^{p+l}$ the modulo operation may be ignored, and the number of cells to the right equals the number of bits in $2^{p+l} - j - 2^p$. This can be related to the number of bits in j, because $2^{p+l} - j$ is $\bar{j} + 1$, with only the first $p + l$ bits taken. Thus the number of cells to the right is the number of bits in $\bar{j} - (2^p - 1)$. Combining this with the fact that $j \leq 2^{p+l} - 2^p$, the total number of bits (and hence cells) to the left and right of the refinement can not exceed $p + l + 1$.

Processor Load in the One-Dimensional Model

It seems to be easiest, at first, to analyze the processor load only for those positions in which each cell of the refinement is assigned to a different processor — remember that by assumption the region of refinement consists only of as many cells as there are processors. Assume that the cells of a grid at level $p+l$ represent the refinement, $1 \leq l \leq p$; they can be labelled by members of a $(p+l)$-bit Gray code sequence.

Lemma 5.3. *For any $l \geq 1$ the Gray code sequence G_{p+l} can be represented as a tensor product of two sequences G_p and G_l as follows:*

$$\begin{aligned}
G_{p+l}(0) &= G_l(0).G_p(0)\\
\vdots \quad & \quad \vdots\\
G_{p+l}(2^p-1) &= G_l(0).G_p(2^p-1)\\
G_{p+l}(2^p) &= G_l(1).G_p(2^p-1)\\
\vdots \quad & \quad \vdots\\
G_{p+l}(2^{p+1}-1) &= G_l(1).G_p(0)\\
\vdots \quad & \quad \vdots\\
\vdots \quad & \quad \vdots\\
G_{p+l}(2^{p+l-1}) &= G_l(2^l-1).G_p(2^p-1)\\
\vdots \quad & \quad \vdots\\
G_{p+l}(2^{p+l}-1) &= G_l(2^l-1).G_p(0),
\end{aligned}$$

where the fullstop denotes concatenation.

Proof. By induction using Definition 2.1 of Gray code sequences.

Definition 5.2. A region of refinement of 2^p cells belonging to a grid at level $p+l$, $l \geq 1$, is in a ***special position*** if the first l bits of all refined cells are identical:

$$\begin{array}{ll}
G_l(j).G_p(0), \cdots, G_l(j).G_p(2^p-1), & j \text{ even}\\
G_l(j).G_p(2^p-1), \cdots, G_l(j).G_p(0), & j \text{ odd}, \quad 0 \leq j \leq 2^l-1.
\end{array}$$

All labels in G_{p+l} having $G_l(j)$ as a prefix belong to the jth *ordinal group* of G_{p+l}.

Remark. A region of refinement in *any* special position satisfies $\mathcal{X}(C_{start}) = 0$ and $\mathcal{X}(C_{end}) = 1$.

A refinement in a special position consists of 2^p cells that are identical in their leftmost l bits $x_0 \cdots x_{l-1}$ and form a Gray code sequence of length 2^p in their remaining p bits. In this case, the mapping of labels to processors amounts to shifting and rotating the Gray code sequence G_p: a refined cell C with label $\mathcal{L}(C) = x_0 \cdots x_{l-1}c_0 \cdots c_{p-1}$ is mapped to processor

$$\begin{aligned}
\mathcal{P}(C) &= \{x_0 \cdots x_{l-1}c_0 \cdots c_{p-l-1}\} \oplus \{c_{p-l} \cdots c_{p-1}0 \cdots 0\}\\
&= \{c_{p-l} \cdots c_{p-1}c_0 \cdots c_{p-l-1}\} \oplus \{x_0 \cdots x_{l-1}0 \cdots 0\}.
\end{aligned}$$

Since this transformation represents a bijective mapping, each refined cell is assigned to a different processor, so each processor contains exactly one cell of the region of refinement.

Now consider the outside cells, and start with the case j even. By assumption, $\mathcal{L}(C_{start}) = x_0 \cdots x_{l-1}0 \cdots 0$ and $\mathcal{X}(C_{start}) = 0$. So, by Lemma 5.1, all cells L to the left of C_{start} have labels of the form $\mathcal{L}(L) = y_0 \cdots y_{l-1}0 \cdots 0$ with $\mathcal{X}(L) = 0$, and therefore the label is identical to the processor id $P(C) = y_0 \cdots y_{l-1}0 \cdots 0$. Since the labels of all cells to the left of C_{start} are distinct, they are mapped to different processors. Similary, $\mathcal{L}(C_{end}) = x_0 \cdots x_{l-1}10 \cdots 0$, and by Lemma 5.2 all cells R to the right of C_{end} have labels of the form $\mathcal{L}(R) = z_0 \cdots z_{l-1}0 \cdots 0$ with $\mathcal{X}(R) = 1$, and again the labels are equal to the processor ids. Thus, all cells outside the refinement have different labels and are therefore mapped to different processors. Consequently, each processor contains at most one outside cell. Alternatively, if j is odd then the above argument applies as well by exchanging the rôles of C_{start} and C_{end}. We have now proved the following

Theorem 5.1. *When the domain contains a region of refinement of 2^p cells belonging to a grid at level $p+l$, $1 \leq l \leq p$, in a special position then each processor contains one refined cell and at most one outside cell.*

The notion of special sequences can be generalized to allow for longer sequences.

Corollary 5.1. *When the domain contains a region of refinement of $m2^p$ cells belonging to a grid at level $p+l$, $1 \leq m \leq 2^l$, $1 \leq l \leq p$, in a special position then each processor contains m refined cells and at most one outside cell.*

Proof. Same as for the previous theorem, except that now every processor contains m refined cells.

Thus, in case of a *complete* refinement the distribution of work is uniform: each processor contains 2^l (refined) cells (and no outside cell). We can now relax our assumptions about the position of the region of refinement within the domain.

Corollary 5.2. *When the domain contains any region of refinement consisting of $m2^p$ cells belonging to a grid at level $p+l$, $1 \leq l \leq p$, then each processor contains at most $m+1$ refined cells and one outside cell.*

Proof. Any sequence of length $m2^p$ can be embedded into a special sequence of length $(m+1)2^p$. Now apply Corollary 5.1.

The corollary states that the maximum load per processor is $m+2$ cells. One can easily find examples for which this bound is achieved (see Figure 2). Since there are m times more refined cells than there are processors, the processor load is within a small constant from being optimal.

Now, we are in the position to drop all assumptions with regard to the region of refinement.

Corollary 5.3. *When the domain contains a region of refinement consisting of at most $m2^p$ cells belonging to a grid at level $p+l$, $1 \leq l \leq p$, then each processor contains at most $m+1$ refined cells and one outside cell.*

Communication Traffic in the One-Dimensional Model

It is assumed that the connection between any two processors in the hypercube is bi-directional, that is, two processors can simultaneously send messages to each other. We also consider each time step to be partitioned into 2^l slots, so that in the jth slot processors belonging to the jth ordinal group can emanate their messages. Since a processor can contain several cells, there is a unique cell to processor assignment in each time slot, and at most one cell per processor can emanate messages in each time slot. This is particularly useful for an implementation on the Connection Machine, where at each time step a processor is either 'inactive' or executes the same instruction as all other active processors (data parallelism). For the following types of information transfer one can show that there is at most one message on any processor-to-processor link:

- Each cell transmits a message to its parent.
  The parent is in the same processor as the cell or in an adjacent processor, and each link is occupied by at most one message.
- Each parent transmits a message to its two children.
  As above.
- Each cell transmits a message to its closest right (left) neighbor.
  If the label of the cell and that of its closest right (left) neighbor are adjacent members in the $(p+l)$-bit binary reflected Gray code sequence, then the cell and its neighbor occupy adjacent processors (because there is only one region of refinement they could not be neighbors in any shorter sequence). Otherwise, assume that the cell C originates at level k and apply Lemmata 5.1 and 5.2 as follows: C sends the message first to an intermediate processor containing C's left (right) neighbor N at level k; this processor in turn forwards the message to the processor holding the right (left) neighbor of N on level $k-1$. For instance, if cell 11001 in Figure 2 wants to send a message to its right neighbor 11010, it first sends it to its left neighbor on level 4, 11000, which in turn forwards it to its right neighbor on level 3, 11010. Because of the bidirectional links, and the fact that the right (left) neighbor of N on level $k-1$ is not its *closest* right (left) neighbor, each link carries at most one message.

Some of the 2^l time slots may not be needed because in the single region of refinement problem, as in any partial hierarchy, not all cells exist.

The Two- and Three-Dimensional Models

One can regard the cells in a two-dimensional domain as a sequence or stack of one-dimensional domains (this is possible because of the interleaved Gray code: for each one-dimensional domain only the coordinates in one direction form a Gray code). Similarly, a three-dimensional domain is viewed as a stack of planes, where each plane consists of a sequence of one-dimensional domains.

In the multi-dimensional case one can distinguish one-dimensional domains in each coordinate. As the labels of the closest neighboring cells can differ in at most two bits in case of one dimension, they can differ in four bits in two dimensions and in six bits in three dimensions.

On a p-dimensional hypercube, each processor contains at most $2(\sqrt{m}+2)$ cells for a two-dimensional region of refinement of $\sqrt{m}2^{p/2} \times \sqrt{m}2^{p/2}$ cells, and at most $3(m^{1/3}+2)$ cells for a three-dimensional region of refinement of $m^{1/3}2^{p/3} \times m^{1/3}2^{p/3} \times m^{1/3}2^{p/3}$ cells. Thus, the workload per processor is linear in the number of dimensions and thus improves with increasing number of dimensions.

6. Experimental Results

This section contains a few experimental results. We have simulated the algorithms to gather information on their behavior and we have also implemented them on the Connection Machine as proof-of-principle.

The simulations were conducted with a one-dimensional model problem consisting of a zone of refinement, a shock for instance, moving left to right. As this region of refinement moves, information is collected on the maximum and minimum processor loads and message distances. Figures 4 and 5 show typical results for the moving refinement.

```
/*
Compute a processor label on the CM. The label has d bits and the
processor dimension is p.  The value of the label is in absolute
bit location v and the result is left in absolute bit location p.
Only on selected processors.
*/
ProcLabel( v, d, pv, p )
CM_memaddr_t v, pv;
int d, p;
{
unsigned int llen;
llen = 2 * p - d;
CM_move( pv + llen, v, (unsigned) (d-p) );
CM_logxor( pv + llen, v+p, (unsigned) (d-p) );
CM_move( pv, v+d-p, llen );
}
```

Figure 3: Paris in c function to perform the mapping of processor labels to processor numbers.

The results in Figure 4 agree with the results in Corollary 5.3, and those in Figure 5 agree with our discussion on the communication traffic. Simulation results for different numbers of processors and refinement region sizes have been obtained; they are very similar to the figures shown here.

To test our thesis that this method can be efficiently implemented on a SIMD architecture, we ran the algorithm on a Connection Machine. The code was written in c using the "Paris in c" functions. Figure 3 gives an impression of the simplicity of the code, in this case the function performed is the processor mapping.

7. Conclusions

In this paper we have presented a static procedure for implementating recursive mesh refinement on hypercubes. The assignment to processors of cells belonging to a hierarchy of adaptive uniform grids is performed in two stages:

1. Each cell is assigned a *unique* label.
2. Each label is mapped to a processor identifier (id).

The first stage preserves coherence: it is easy to find the labels of siblings, ancestors and descendants of a cell in the hierarchy of grids; once the label has been determined the corresponding processor can be found without much effort. The second stage makes it possible to ensure that related cells are allotted to physically close processors of the hypercube, and that the work load is distributed reasonably across all processors for our applications. Since the labelling strategy is static, a cell can determine the label of any other cell at any level without requiring external information.

For the solution of PDEs on multi-dimensional grids, we advocate the use of an *interleaved* Gray code that does not represent the bits associated with a coordinate as a contiguous group, but 'scatters' or interleaves the bits associated with all coordinates. For practical applications, interleaved Gray codes result in essentially constant communication times.

Our scheme has the following advantages:

- It does not require any prior knowledge on the maximal level of refinement.
- It is predictable: the processor assignment for *any potential* cell can be determined in advance.

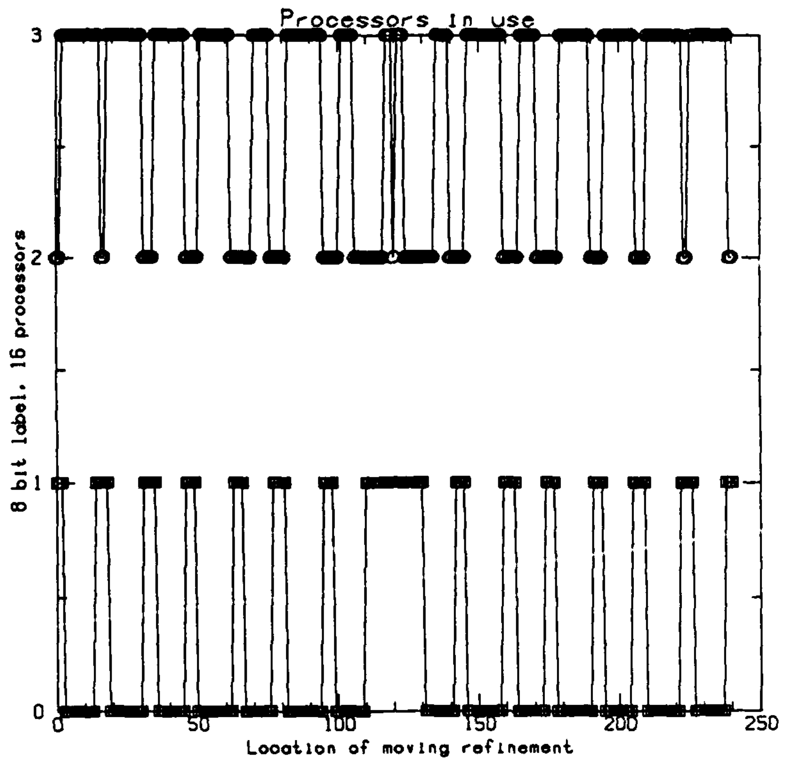

Figure 4: Processor load for 16 processors and an 8 bit label. The refinement is 16 cells long. The top line is the maximum number of cells per processor; the bottom line is the minimum number of cells per processor.

- It is simple and fast: each cell can determine the processor assignment of any other by simple bit-operations.
- It is consistent: in case of *complete* refinement the assignment of cells to processors is uniform.
- It performs well in the case where a few regions require thorough refinement.
- The load balance improves with increasing dimensionality of the problem.
- It takes full advantage of the rich hypercube processor interconnection topology.
- The theoretical results are corroborated by experimental evidence.

Future work will deal with dynamic processor assignment strategies that require more than local knowledge, and processor assignment strategies for many levels of refinement ($l > p$).

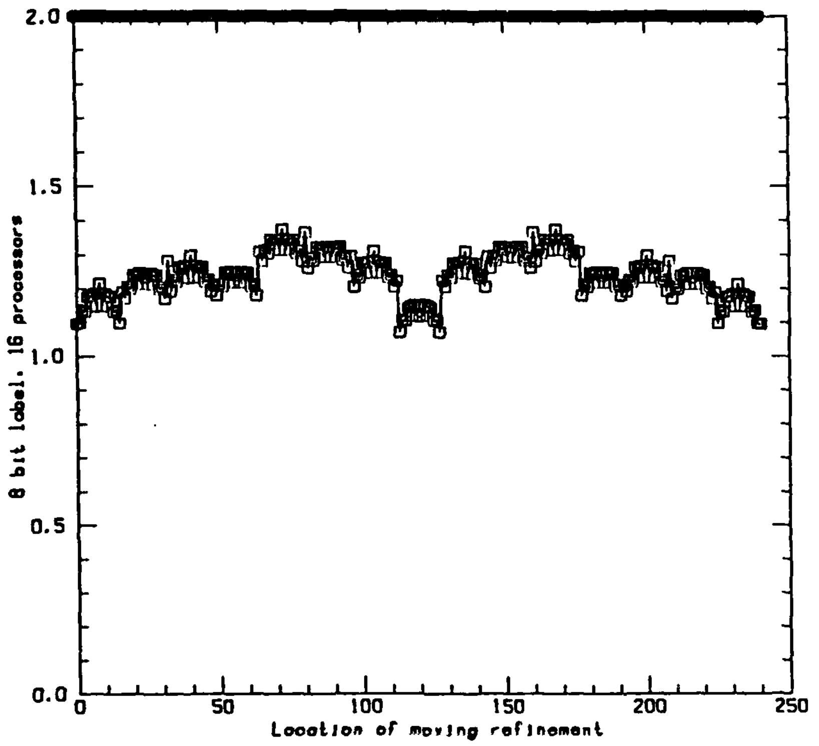

Figure 5: Message distances for the the same case as in Figure 4. The top line is the maximum message distance and the bottom line is the average message distance (the minimum is always 1).

References

[1] Gilbert, E.N., *Gray Codes and Paths on the N-Cube,* The Bell System Technical Journal, (May 1958).

[2] Gropp, W.D., *Local Uniform Mesh Refinement for Elliptic Partial Differential Equations,* Research Report 278, Dept. of Computer Science, Yale University, 1983.

[3] ———, *Local Uniform Mesh Refinement on Loosely Coupled Parallel Processors,* Research Report 352, Dept. of Computer Science, Yale University, 1984.

[4] Li, S.-X. and Loew, M.H., *The Quadcode and Its Arithmetic,* CACM, 30 (1987), pp. 621–6.

[5] Saad, Y. and Schultz, M.H., *Some Topological Properties of The Hypercube Multiprocessor,* Research Report 389, Dept Computer Science, Yale University, 1984.

[6] Wu, A.Y., *Embedding of Tree Networks into Hypercubes,* Jour. Par. Distr. Comp., 2 (1985), pp. 238–49.